VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Plánování vyskladňování zásilek knižního skladu

Planning of Picking Deliveries for a Library Store

2014 Vojtěch Kotík

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Vojtěch Kotík**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Plánování vyskladňování zásilek knižního skladu
Planning of Picking Deliveries for a Library Store

Zásady pro vypracování:

Cílem práce je vytvořit informační systém pro knižní sklad, jehož součástí bude automatické plánování cesty pro vyskladňování zásilek.

Práce bude splňovat následující body:
1. Analýzu podnikového prostředí, kde by systém měl být nasazen.
2. Analýzu a návrh informačního sysému.
3. Implementaci standardních součástí informačního systému.
4. Návrh a implementace modulu pro definici rozložení skladu.
5. Návrh a implementace algoritmu pro plánování efektivní cesty skladníka pro vyskladnění zásilky.
6. Srovnání s existujícími aplikacemi.

Seznam doporučené odborné literatury:

Podle pokynů vedoucího bakalářské práce.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Petr Lukáš**

Datum zadání: 01.09.2013

Datum odevzdání: 07.05.2014

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 7. května 2014 .....

Rád bych poděkoval vedoucímu bakalářské práce Ing. Petru Lukášovi za odbornou pomoc a konzultaci při vytváření této práce.

Abstrakt

Základním logistickým pilířem velko i maloobchodu jsou sklady. Moderní sklad se při svém provozu neobejde bez kvalitního informačního systému, který zajistí urychlení a zpřehlednění skladových procesů. Systém je nutno většinou přizpůsobit specifickým problémům konkrétního prostředí a skladovanému sortimentu. Tato bakalářská práce se zabývá zmapováním procesů knižního skladu a jejich převedením do uceleného informačního systému. Vzhledem k rozsáhlosti problematiky je tato práce orientována na běžné procesy spojené s pohyby zboží a souvisejícími problémy.

Klíčová slova: informační systém, sklad

Abstract

Warehouses are important part of wholesale and retail logistics. Modern warehouse requires for regular operations high-quality information system which will ensure fast and smooth processes. The system must be usually adapted to handle specific problems of particular environment and stored goods. This bachelor thesis is about describing processes of book warehouse and transforming them into information system. With regards to fact that field of this subject is wide this thesis is focused on common processes associated with movement of goods and related problems.

Keywords: information system, warehouse

Obsah

1 Úvod

Základním logistickým pilířem velko i maloobchodu jsou sklady. Moderní sklad se při svém provozu neobejde bez kvalitního informačního systému, který zajistí urychlení a zpřehlednění skladových procesů. Systém je nutno většinou přizpůsobit specifickým problémům konkrétního prostředí a skladovanému sortimentu. Tato bakalářská práce se zabývá zmapováním procesů knižního skladu a jejich převedením do uceleného informačního systému. Vzhledem k rozsáhlosti problematiky je tato práce orientována na běžné procesy spojené s pohyby zboží a souvisejícími problémy.
V kapitole 2 si nejprve rozebereme a zanalyzujeme podnikové prostředí. V další kapitole bude podrobně popsán návrh informačního systému a databáze. Následovat bude návrh komponenty pro design skladu. Poslední část je věnována výpočtu cesty skladníka.
Součástí práce je i funkční prototyp aplikace, který lze nalézt na přiloženém DVD.

2 Analýza podnikového prostředí

Základem je **sklad**, který poskytuje příjem, skladování a výdej knih a souvisejícího sortimentu. Kromě procesů provázejících běžné provozní operace musí systém podporovat i různé podpůrné procesy skladového managementu. Sklad je fyzicky rozdělen do sektorů, které se dále dělí na jednotlivá umístění. Pro urychlení práce jsou používány čárové kódy k jednoznačné identifikaci knih, skladových umístění i přepravních obalů.

2.1 Statická analýza

Sortiment

Představuje knižní tituly vedené ve skladu. U každého sortimentu je potřeba sledovat aktuálně dostupné množství a trendy v objednávkách, což umožní vysokou dostupnost velkého počtu položek sortimentu.

Přepravní obaly

Pro pohyb sortimentu po skladu a dopravu k zákazníkům jsou využívány dva typy přepravních obalů. Znovupoužitelné plastové bedny typu Integra jednotné kapacity a kartonové obaly, které se dají přizpůsobit na míru. Plastové bedny mají své nálepky s čárovým kódem. Při specifických činnostech, jako je přeprava k zákazníkovi je přidáván ještě doplňkový čárový kód (identifikace zásilky) na dodatkové nálepce.

Sektory

Sklad je pro přehlednost rozdělen do sektorů. Každý sektor má své unikátní označení, které je generováno z rostoucí číselné řady, což umožňuje snadnou orientaci ve skladu.

Skladová umístění

Sektory jsou dále rozděleny na jednotlivá skladová umístění, která slouží pro skladování sortimentu. Knihy jsou zde naskladňovány skladníky a poté vyskladňovány chystači. Čárový kód skladového umístění (zobrazen na obrázku 1) je složen z kódu sektoru, regálu a police.
Skladová umístění jsou trojího typu.
Regálové (viz obr. 2) se skládá ze čtyř polic stejných rozměrů. Vzhledem k proměnné velikosti a objemu jednotlivých titulů může být počet naskladněných položek různý.
Paletové (viz obr. 3) jsou určena pro rozměrný sortiment a rychloobrátkové tituly, kterých jsou objednány velké počty.
Krabicové regály (viz obr. 4) umožňují naskladnit sortiment, který by v klasických regálech způsoboval problémy, jako jsou tenké knížky bez textu na hřbetu. V jednom umístění může být vždy jen jedna kartonová krabice obsahující stejný titul.

Zákazníci

Fyzické nebo právnické osoby objednávající zboží. U každého zákazníka musíme mít k dispozici adresu pro doručení objednaného sortimentu.

Obrázek 1: Čárový kód umístění

Obrázek 2: regálové umístění

Obrázek 3: paletové umístění

Obrázek 4: Krabicové umístění

Objednávky

Podklad pro dodávku sortimentu zákazníkům. Každá objednávka se skládá z položek sortimentu a objednaného množství.

Zaměstnanci

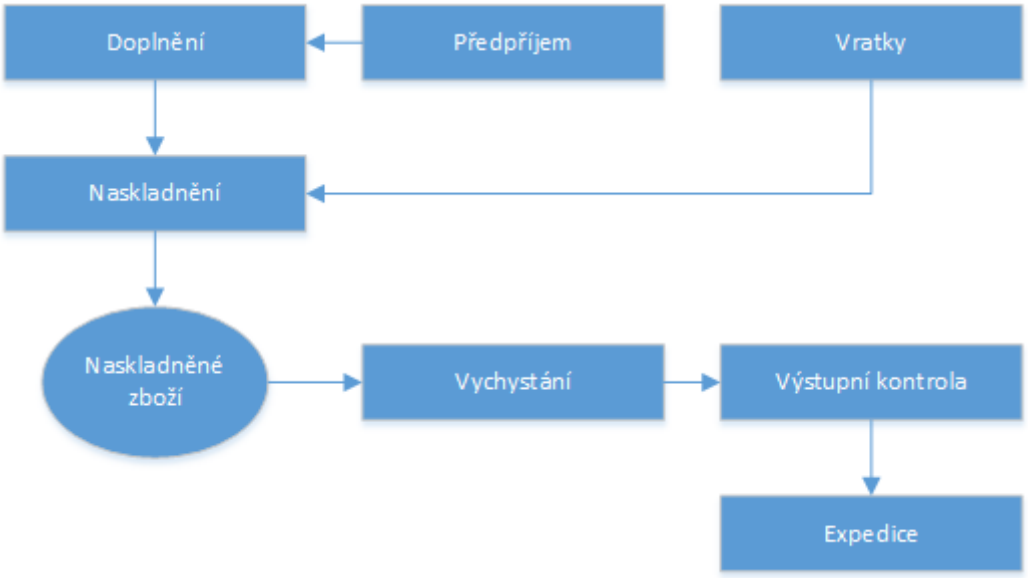
Systém rozlišuje několik typů uživatelů. **Chystači** mají na starosti pouze vychystávání zboží. Na této pozici pracují hlavně brigádníci. **Skladníci** jsou zodpovědní za pokročilejší operace se sortimentem, při kterých používají PDA. **Vedoucí skladu** zpracovává objednávky a dohlíží na provoz skladu. **Administrátor systému** se stará o správu skladového IS.

2.2 Dynamická analýza

Procesy běžného provozu

- Příjem zboží
  - Předpříjem
  - Tvorba doplnění
  - Zpracování zákaznických vratek
  - Naskladnění
- Výdej zboží
  - Vychystávání podle objednávek

  - Výstupní kontrola
  - Expedice zásilek
- Vnitřní pohyby
  - Přeskladnění
  - Inventura
  - Čištění skladových umístění



Obrázek 5: Procesy provozu skladu

Předpříjem
Hrubá příprava zboží po vyložení přepravního kontejneru. Skladníci provádějí vizuální kontrolu zboží a zadávají do systému dosud neznámé knihy včetně rozměrů, váhy a dalších důležitých atributů.

Tvorba doplnění
Příprava zboží pro naskladnění. Skladník příjmu zboží naskládá do krabic nebo na palety. Po načtení čárového kódu zboží a zadání počtu dojde k vygenerování optimálního skladového umístění pro naskladnění a vytištění průvodní nálepky.

Zpracování zákaznických vratek
Zákazníci vracejí neprodané nebo poškozené zboží prostřednictvím vratek. Skladníci kontrolují jakost vráceného sortimentu a načítají jej do systému. Systém vybere optimální

skladové umístění a vytiskne lístek, který skladník vloží do knihy a dá ji na pásový dopravník. Jiný skladník odpovídající za rozsah sektorů, ve kterém se nachází nově vybraný sektor, knihu z dopravníku odebere a umístí ji do odpovídající bedny, která je po zaplnění odvezena k naskladnění.

Naskladnění

Skladníci používají PDA se čtečkou čárových kódů, které komunikují se skladovým IS přes wifi . Po fyzickém umístění zboží skladník načte čárový kód z nálepky vytvořené na příjmu a kód cílového umístění.

Vychystávání zboží podle objednávek

Chystač si na „pick station“ vytiskne *pick* – položky objednávky k vychystání. Poté se pohybuje po skladu, kde sbírá jednotlivé knihy, umisťuje je do přepravní bedny a lepí na ně nálepky, aby bylo možné snadno zjistit, jaký sortiment se ve které přepravní bedně nachází v jakém množství (viz obrázek 6). Používá přitom vozíček složený z podvozku a dalších přepravních beden (pro ilustraci viz obrázek 7).

Obrázek 6: Nálepka na knihu

Vychystávání může být ukončeno na jiné pick station. Při ukončení vychystávání chystač edituje položky, která na umístění nebyly nalezeny, byly poškozené nebo se již do bedny nevešly. Po ukončení editace a odložení bedny pokračuje stejným způsobem s dalším pickem. Vychystané přepravní bedny jsou takto připraveny k ověření výstupní kontrolou a následné expedici zákazníkům.
Objednávka vytvořená zákazníkem zpravidla obsahuje velký počet položek sortimentu. Spousta položek je naskladněna na několika umístěních po celém skladu. Celá objednávka se většinou nevejde do jediné bedny a proto je nutno ji rozdělit. Chystači svou cestu skladem začínají a končí na předem určených místech, tzv. „pick station“. Úkolem informačního systému je vypočítání optimální varianty rozdělení objednávky a určení nejvhodnějších sektorů, což vede k minimalizaci vzdálenosti nachozené chystači a maximalizaci efektivity celého procesu vychystávání. S vychystáváním souvisí jeden z důležitých bodů této práce – návrh optimální cesty skladníka (viz kapitola 7)

Obrázek 7: Chystací vozíčky

Výstupní kontrola

Výstupní kontrolor provádí vizuální kontrolu vychystaných knih, tj. sortimentu, který odpovídá objednávkám připraveným k expedici. Jednotlivé položky postupně načítá pomocí čárových kódů do systému, který provádí párování s položkami odpovídajícího picku. Účelem je odhalit defekty a případné záměny chystačem za jiný titul. Po dokončení kontroly systém vytiskne dodací list a přepravní nálepky. Kontrolor zásilku zabalí a pošle po pásovém dopravníku na expedici.

Expedice zásilek

Stará se o balení zásilek na palety a naložení do správného auta. U zásilek pro poštu a soukromé přepravní společnosti provádí import dat do jejich interních systémů pro usnadnění manipulace za použití vlastních přidělených číselných řad pro balíky.

Přeskladnění

Přesun zboží z jednoho skladového umístění na jiné. Jedním z důvodů je potřeba uvolnění skladového umístění přeskladněním posledních několika kusů. Provádí skladník s PDA.

Čištění skladových umístění

Optimalizace využití skladu snášením stejných titulů na co nejmenší počet umístění.

Inventura

Jednou ročně je prováděna inventura, při které dochází k porovnávání informací o skladových zásobách s reálnými počty na umístěních. Účelem je zjistit chybějící/přebývající zboží a uvést informace v systému do korektního stavu. Počítání je provedeno opakovaně různými lidmi pro větší přesnost.

3 Datová analýza

Pro implementaci databáze jsme zvolili SŘBD Microsoft SQL Server 2012. Půjde tedy o standardní relační databázi s využitím rysů objektově relačního rozšíření. Celé schéma databáze včetně názvů sloupců a jejich datových typů je možné nalézt na přiloženém DVD (příloha A).

Evidence zaměstnanců

Při přijetí nového zaměstnance je prvním krokem vložení jeho osobních údajů do tabulky *Person*. Ta obsahuje jména a kontakty. Ke každému záznamu je navázán záznam z tabulky *Address*, ve které je uchována adresa bydliště.
Dalším krokem je vložení zaměstnance do tabulky *Employee*. Zaměstnanci je vygenerováno osobní číslo, nastaveno datum zahájení pracovního poměru, pracovní pozice a přidělen osobní čárový kód. Tabulka dále obsahuje volitelný atribut konce pracovního poměru. Ten slouží pro zamezení přístupu zaměstnancům po odchodu z firmy. V případě smazání záznamu bychom přišli o informace, které činnosti tento zaměstnanec vykonal. Dalším volitelným parametrem je heslo, kterým může být zabezpečen přístup k pokročilejším funkcím systému.
Tabulku *Employee* doplňuje číselník *Positions*, který uchovává pracovní pozice. Záznamy jednotlivých pozic se skládají z jejich zobrazovaného jména, volitelného popisu a atributu *skill*, který určuje úroveň odbornosti dané pozice.

Relační schéma je uvedeno na obrázku 8 .

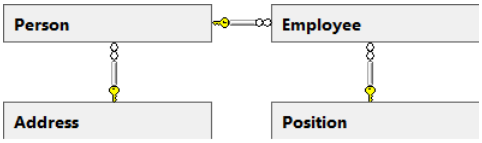
Evidence zákazníků

Zákazníci jsou uloženi v tabulce *Customer*. Systém rozlišuje firmy a fyzické osoby pomocí atributu type. Tabulka dále obsahuje běžné informace o zákazníkovi, jako je jméno (kontaktní osoba u firmy), bankovní spojení, u firmy IČ a DIČ. Jsou také obsaženy atributy, zda si zákazník přeje dostávat newsletter. Zákazníkova adresa je uchována v tabulce *Address* identickým způsobem jako u zaměstnance.

Relační schéma je uvedeno na obrázku 9 .

Katalog sortimentu

Každé zboží musí být před naskladněním zavedeno do tabulky *Product*. Z číselníku *ProductType* vybereme typ produktu. Následně vyplníme základní údaje, jako je ISBN, čárový kód, název zboží, informace o autorovi a nakladateli. Tabulka obsahuje také váhu, rozměry a automaticky vypočítaný objem, což jsou nezbytné parametry pro efektivní plánování skladování a vyskladnění.



Obrázek 8: Evidence zaměstnanců

Obrázek 9: Evidence zákazníků

Skladová umístění

Sklad je rozdělen do dvou úrovní – sektory a jednotlivá umístění v nich. Sektory jsou uchovávány jako záznamy tabulky *Sector*. Číselník *SectorType* obsahuje typy sektorů. Tabulka *Placement* obsahuje regály a police v sektorech (viz kapitola 2.2).

Evidence naskladněného sortimentu a jeho pohybu

Informace o tom, kde je zboží naskladněno a v jakém množství jsou uloženy v tabulce *ProductPlacement*. V každém jejím záznamu je uloženo id produktu, umístění, aktuální dostupný počet a doplňkové atributy, pomocí kterých je možno zabránit skladovým operacím s nastaveným počtem zboží.
Při přeskladnění jsou zapisovány záznamy do tabulky *ChangeOfPlacement*, která slouží k uchování historie pohybu zboží. Tabulka je doplněna číselníkem stavů přeskladnění, který zajišťuje konzistentnost a umožňuje například snadno zjistit počet dosud nezpracovaných úkolů.

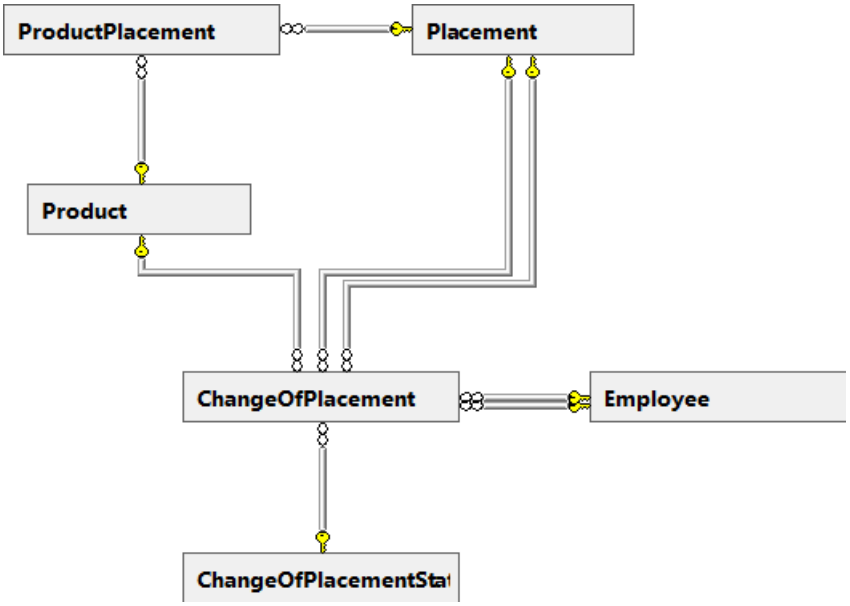
Relační schéma je uvedeno na obrázku 12 .

Přepravní obaly

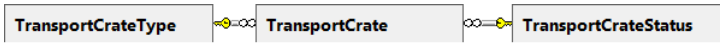
U přepravních obalů je pro efektivní vychystávání nutno mít k dispozici jejich maximální vnitřní objem a nosnost. K tomu slouží číselník *TranportCrateType*. Tabulka *TransportCrate* obsahuje záznamy o jednotlivých obalech. Její atribut status umožňuje zjistit, zda obal v danou chvíli něco obsahuje, nebo je možno jej použít. Hodnoty stavu jsou opět uchovány v odděleném číselníku.

Obrázek 10: Katalog sortimentu

Obrázek 11: Skladová umístění



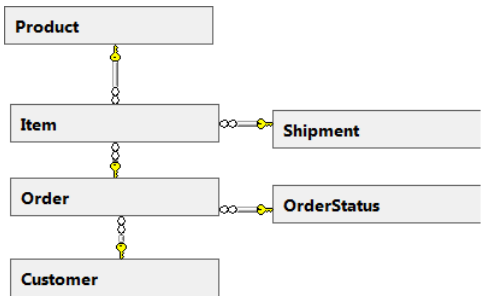
Obrázek 12: Evidence naskladněného sortimentu a jeho pohybu



Obrázek 13: Přepravní obaly

Relační schéma je uvedeno na obrázku 13 .

Objednávky

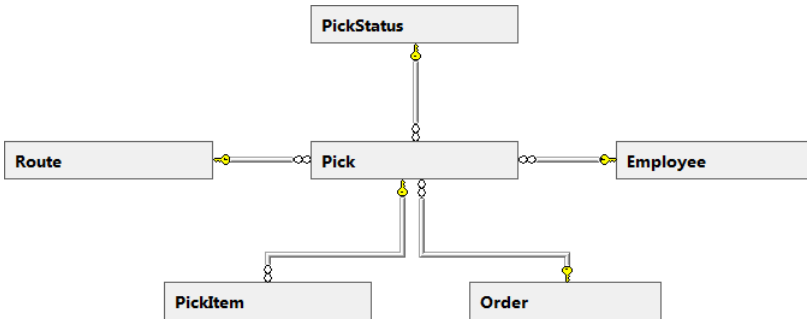


Obrázek 14: Objednávky

Jádrem části systému, který se stará o zpracování objednávek zákazníků je tabulka *Order*. Obsahuje důležité informace o tom, kterému zákazníkovi daná objednávka patří, kdy byla přijata a termín, do kterého musí být realizována. U každé objednávky je zaznamenán její stav za pomoci číselníku *OrderStatus*. Adresář zákazníků je uložen v tabulce *Customer*, která obsahuje hlavně kontaktní informace. Každému zákazníkovi je přiřazen jeden záznam tabulky *Address* pro účely doručení objednaného zboží. Pro položky objednávek je připravena rozkladová tabulka *Item*.

Relační schéma je uvedeno na obrázku 14 .

Vychystávání



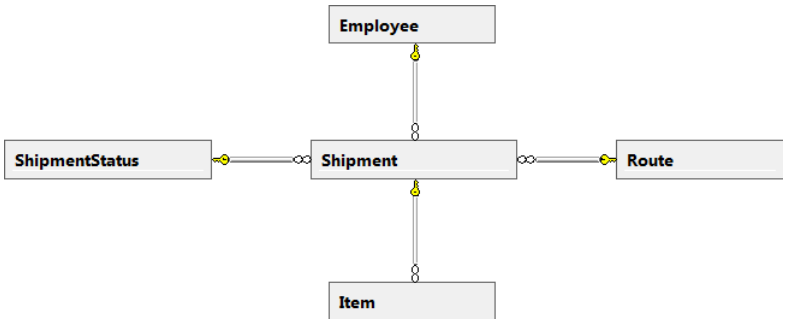
Obrázek 15: Vychystávání

Picky slouží pro vychystání objednávek a uchování historie. U každého picku jsou uchovány jednotlivé položky a informace, ze kterého skladového umístění má chystač

zboží vychystat a v jakém počtu. Číselník *Route* umožňuje dělení picků podle distribučních tras a jejich zpracování ve stanoveném pořadí. Pomocné atributy času zahájení a ukončení vychystávání umožňují sledovat výkonnost jednotlivých chystačů.

Relační schéma je uvedeno na obrázku 15 .

Expediční zásilky



Obrázek 16: Expediční zásilky

Výstupní kontrola přebalí vychystané picky do přepravních zásilek, které odešle na expedici. Po kontrole dojde k aktualizaci tabulky *Item*, kde je k položkám objednávky doplněn atribut expediční zásilky pro snadné dohledání, ve které zásilce se daná položka nachází.

Relační schéma je uvedeno na obrázku 16 .

4 Architektura informačního systému

Tato část práce je věnována popisu vrstvené architektury systému a jeho modulárnosti. První část popisuje datovou vrstvu a objektově relační mapování. Druhá část se věnuje popisu modulů systému. Kompletní zdrojové kódy jsou k dispozici na přiloženém DVD (příloha B).
Systém používá vrstvenou architekturu (viz obrázek 17). Základem je oddělená datová vrstva, která se stará o převody mezi relační databází a objektově orientovaným programovacím jazykem.
Aplikace je rozdělena do oddělených tematických modulů shodné architektury, což umožňuje její snadnou rozšiřitelnost do budoucna.

Datová vrstva

Pro přístup k databázi je použito objektově relační mapování. Slouží pro usnadnění práce s databází pomocí zapouzdření SQL příkazů a převod dat do objektově orientovaného programovacího jazyka. Současně ale neznemožní klasický přístup k databázi, což je možno využít pro jednoduché příkazy, u kterých by bylo použití ORM kontraproduktivní.

Třída Session

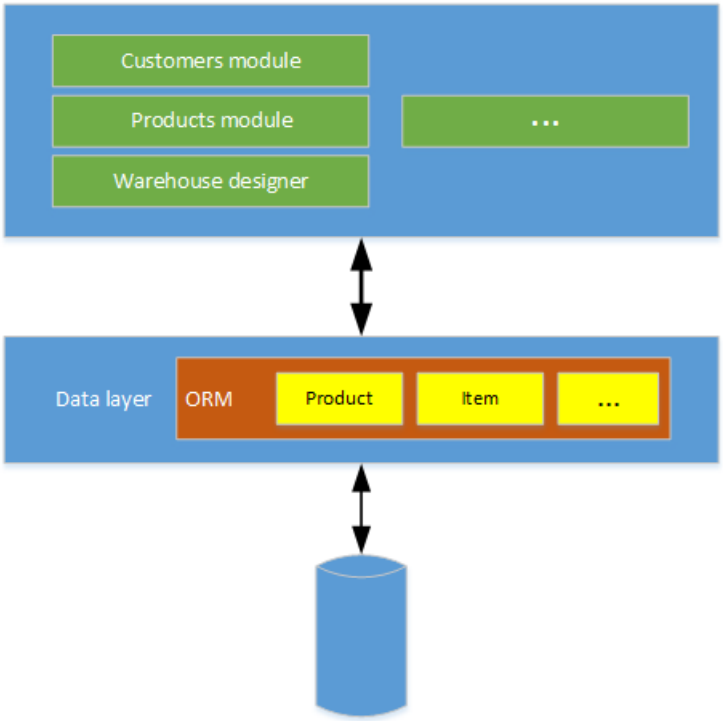
Statická třída tvořící centrální přístupový bod k databázi. Umožňuje zjistit informace o aktuálním připojení k databázi a probíhající transakci. Pomocí metody *InitializeConnection* umožňuje otevřít připojení k databázi. Metoda *GetLastIdentity* umožňuje zjistit poslední hodnotu automaticky generovaného primárního klíče zadané tabulky. Jednotlivé příkazy připravuje metoda *PrepareCommand*, která nastaví údaje o připojení a vrátí připravený příkaz.

Třída DataObject

Abstraktní třída sloužící pro obecné definování chování ORM objektu. Tyto objekty slouží jako nosné pro data (někdy nazývané Doménové objekty), ale obsahují rovněž metody pro práci s nimi v rámci jedné třídy. Definuje metody, které by měly být v odvozených třídách překryty. Pro uchování většího počtu instancí objektu *DataObject* slouží třída *DataObjectSet*. Umožňuje hromadné čtení a zápis dat do databáze, čehož je využito například při práci s položkami objednávky.

Datové objekty

Pro každou databázovou tabulku je vytvořena samostatná třída dědící z třídy *DataObject*, která slouží pro veškeré operace nad touto tabulkou. Tato třída obsahuje atributy pro uložení dat zpracovávaného záznamu. Dále je obsažena ještě druhá sada identických atributů pro uchování originální dat v případě jejich změn. Metody pro práci se záznamem jsou rozděleny do několika skupin. Pro ilustraci můžeme na obrázku 18 vidět, co vše se může nacházet v datovém objektu pro modelový případ - zákazník.



Obrázek 17: Architektura informačního systému

Customer
+IdCustomer : int +FirstName : string +LastName : string -Company : int -originalFirstName : string -originalLastName : string -originalCompany : int
+IsChanged() : bool +LoadRecord(in SqlDataReader) +PrepareCommandParameters() +InsertToDatabase() +UpdateDatabase() +DeleteFromDatabase() +CopyTo(in customer) +Load(in idCustomer : int) : Customer +LoadByCompany(in company : int) : DataObjectSet +LoadAll() : DataObjectSet +LoadBy(in where : string) : DataObjectSet +Exists(in idCompany : int) : bool +Create() : DataObjectSet

Obrázek 18: Ukázková tabulka Customer

První část tvoří instanční metody pro základní databázové operace zprostředkovávající vytvoření, aktualizaci a mazání záznamů. O přetypování databázových datových typů na datové typy objektově orientovaného jazyka se stará metoda *LoadRecord*. Zpětné vkládání dat do parametrů SQL příkazů a jejich převod na odpovídající datové typy provádí metoda *PrepareCommandParameters*. Metoda *IsChanged* umožní zjistit, zda došlo ke změně dat pomocí porovnání aktuálních a originálních hodnot atributů. Vkládání nových záznamů probíhá prostřednictvím metody *InsertToDatabase*. K aktualizaci záznamů slouží metoda *UpdateDatabase*, která připraví změněné atributy do jednoho příkazu a poté provede jejich update. Mazání zajišťuje metoda *DeleteFromDatabase*, která smaže záznam podle primárního klíče aktuálního záznamu. Sadu doplňuje metoda *CopyTo*, která umožňuje jednoduché zkopírování hodnot záznamu do jiného objektu stejného typu.

Pro přehledné načítání záznamů slouží druhá sada tentokrát statických metod. Základní metoda *Load* vrátí jeden záznam tabulky odpovídající primárnímu klíči předanému v parametru. Záznamy je také možno načítat pomocí všech ostatních atributů, které jsou součástí cizích klíčů. K tomuto účelu jsou připraveny specializované metody *LoadBy* + název atributu (cizího klíče), které vracejí kolekci záznamů (*DataObjectSet*) odpovídajících zadané hodnotě. Pro použití vlastních podmínek filtrování lze použít přetíženou metodu *LoadBy*, které předáme pomocí parametru hodnoty, podle kterých vyfiltruje záznamy. V případě nutnosti načtení všech záznamů tabulky lze použít metodu *LoadAll*. Sadu uzavírá metoda *Exists*, která slouží pro ověření existence záznamu podle dané hodnoty primárního klíče.

Nové instance datových objektů by se neměly tvořit standardním způsobem (voláním konstruktoru), ale statickou metodou *Create*, která kromě toho, že vytvoří instanci, zajistí také načtení výchozích hodnot atributů.

Vzhledem k množství tabulek v databázi bylo efektivnější vytvořit generátor mapování. Tento generátor mapování jsme vytvořili s pomocí vedoucího práce.

Krátká ukázka práce s mapováním je k dispozici ve výpisu 1 .

```
{
  // Vytvoreni zaznamu
  Customer customer = Customer.Create();
  customer.FirstName = "Frantisek";
  customer.LastName = "Novak";
  customer.InsertToDatabase();

  // Nacteni zaznamu
  Customer customer = Customer.Load(10);
  customer.Company = 12;
  customer.UpdateDatabase();
}
```

Výpis 1: Ukázka práce s mapováním

Třída Common

Ne vždy je vhodné načítat celé záznamy prostřednictvím objektově-relačního mapování. V mnoha případech se potřebujeme pouze doptat např. na jméno zaměstnance podle hodnoty primárního klíče. Pro tyto jednoduché operace byla vytvořena statická třída *Common*, která obsahuje metody jako *GetEmployeeName(int idEmployee), GetSectorName(int idSector)* . Výhodou tohoto přístupu je, že pokud bychom se např. později rozmysleli, že celé jméno zaměstnance bude kromě jména a příjmení tvořit také titul, stačí provést úpravu na jediném místě v kódu.

Třída Utils

Statická třída, jejímž hlavním účelem je naplnění rozbalovacích nabídek (comboboxů). Záznamy často vyžadují nastavení hodnoty z číselníku, který obsahuje malý počet hodnot a dá se předpokládat, že nebude docházet k jejich častému přidávání či odebírání. Ukázkovým případem jsou stavy objednávek.
Pro jednoduché nastavení hodnoty těchto atributů jsou použity rozbalovací nabídky. Pro jejich naplnění slouží metoda *FillComboBox*, které předáme jako parametr objekt rozbalovací nabídky typu ComboBox, jehož hodnoty chceme naplnit, SQL SELECT příkaz pro načtení hodnot z číselníku a na závěr nastavíme, který parametr bude zobrazován a který obsahuje identifikátor.

5 Aplikační logika

Při návrhu aplikační logiky byla použita modulární architektura. Systém je díky tomu možno do budoucna snadno rozšířit o nové moduly a funkce. Rovněž zjednodušuje správu systému a umožňuje svázání logiky do oddělených tematických celků.

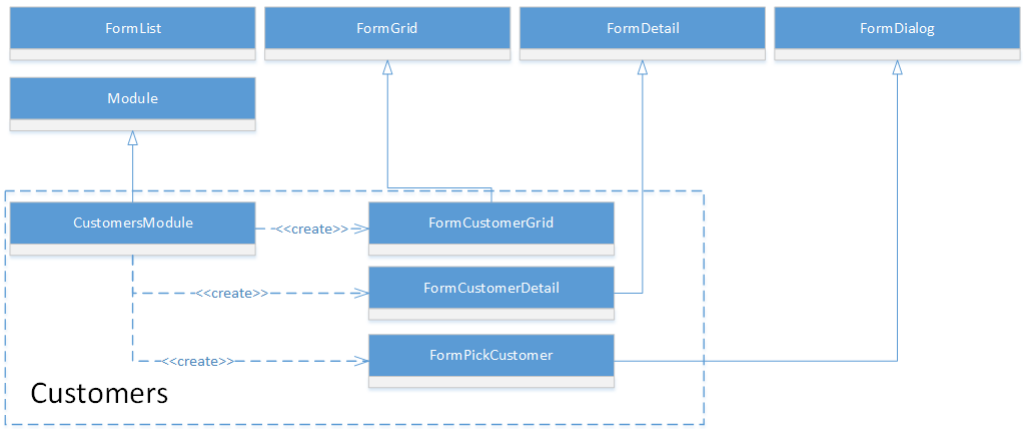
Modulem v aplikaci se rozumí soubor tříd, které slouží k editaci jedné nebo více databázových tabulek. Některé moduly manipulují s daty pouze z jedné tabulky, ale existují jiné moduly (jako např. objednávky), které se starají zároveň o objednávky a zároveň o jejich položky.
V aplikaci rozlišujeme dva typy modulů: (1) needitovatelný přehled + detail záznamu a (2) číselník. Needitovatelný přehled společně s detailem se používá zejména pro CRUD (Create, Read, Update a Delete) operace nad tabulkami, jejichž obsah se průběžně dynamicky mění. Jedná se například o seznamy zákazníků, evidence objednávek, evidenci sortimentu. Moduly typu číselník slouží pro úpravu jednoduchých tabulek, které se často skládají pouze z primárního klíče a popisného názvu a jejichž obsah se v průběhu času příliš nemění.
Pro zajištění konzistentního chování uživatelského rozhraní byly navrženy šablony (abstraktní třídy) pro formulář s needitovatelným přehledem (FormGrid, viz kapitola 5.1), formulář s detailem (FormDetail, kap. 5.2), formulář s číselníkem (FormList, viz kapitola 5.3) a formulář pro výběr záznamu (FormDialog, kap. 5.4). Tyto obecné formuláře obsahují připravené ovládací prvky např. pro založení záznamu, uložení záznamu, obnovení obsahu, atd., přičemž tyto prvky volají abstraktní metody, jejichž chování je přepsáno v konkrétních poděděných třídách.
Pro každý modul je navíc připravena třída zapouzdřující instanciování jednotlivých formulářů modulu. Ukázkový diagram modulu pro evidenci zákazníků vidíme na obrázku 19. Lze si také všimnout, že tento modulární návrh vychází z návrhového vzoru továrna.

5.1 FormGrid

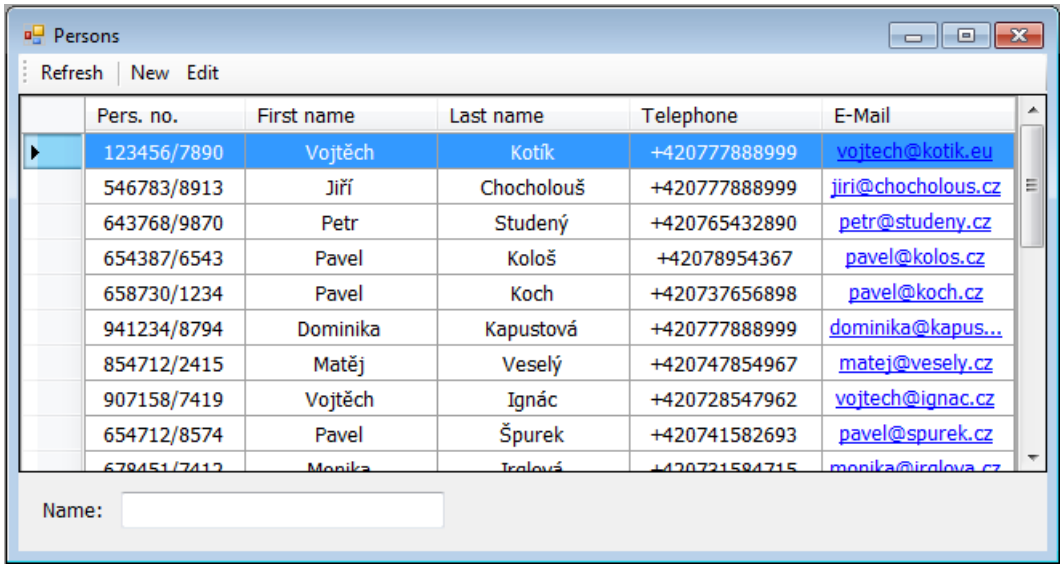
Slouží jako přehled dat daného modulu. V tabulce přehledně zobrazuje záznamy, které je možno filtrovat pomocí vyhledávacího pole. Pro každý *FormGrid* je připraven databázové databázový pohled, který připraví záznamy ve formě vhodné k zobrazení, včetně případného načtení hodnot ze souvisejících tabulek provázaných cizími klíči.

Šablona obsahuje abstraktní metody pro základní operace. Mezi nejdůležitější patří metoda *ApplyFilter*, která vyfiltruje záznamy odpovídající zadané podmínce. Tato metoda je automaticky volána vždy při otevření formuláře nebo při klepnutí na tlačítko *Refresh* .
Editace záznamu je vyvolána dvojklikem na příslušný řádek a obsloužením události *DataGridViewCellDoubleClick* nebo klepnutím na tlačítko *Edit*. Dojde k zavolání metody *EditRecord*. Klepnutím na tlačítko *New* dojde k volání metody *NewRecord*. V obou metodách *EditRecord* i *NewRecord* dojde prostřednictvím třídy modulu k instanciování příslušného formuláře detailu dědícího z *FormDetail* (viz kapitola 5.2).

Ukázkový formulář je na obrázku 20 .



Obrázek 19: Ukázkový modul



Obrázek 20: Ukázkový FormGrid

5.2 FormDetail

Šablona *FormDetail* definuje základní chování a vzhled formulářů pro editace atributů jednotlivých záznamů. Obsahuje obrázkové menu uložené v kontejneru *ToolStrip*, které umožňuje založení nového záznamu, uložení změn, smazání a tisk aktuálně otevřeného záznamu.
Každý formulář dědící z této šablony obsahuje ovládací prvky nutné k editaci jednoho záznamu. Po otevření formuláře voláme metodu *OpenRecord*, které předáváme hodnotu primárního klíče (pokud editujeme záznam) nebo hodnotu *null* v případě, že chceme založit záznam nový. Po načtení záznamu do mapovacího objektu (resp. instanciování objektu nového záznamu) je nutné přenést data do těchto ovládacích prvků. O toto se stará vždy metoda *BindData* .
Provede-li uživatel změnu v některém z ovládacích prvků, dojde k volání příslušné události (u komponenty TextBox se např. jedná o událost *TextChanged*). Ošetřením této události se data vracejí zpět do instance mapovacího objektu.
Nakonec má uživatel vždy 3 možnosti: (1) potvrzení editace, (2) odstranění záznamu, (3) storno editace. V případě potvrzení editace voláme metodu formuláře *SaveRecord*, která následně na mapovací objekt zavolá *InsertToDatabase* nebo *UpdateDatabase*, podle toho, zda jsme vytvářeli nový záznam nebo editovali stávající. V případě odstranění záznamu je volána metoda formuláře *DeleteRecord*, která následně zavolá *DeleteFromDatabase* na instanci mapovacího objektu. Storno editace vede rovnou k uzavření formuláře.
Použití běžného formuláře detailu jako je např. FormCustomerDetail z modulu zákazníků je ilustrováno vývojovým diagramem na obrázku 22 .
Výhoda tohoto přístupu je, že dokud uživatel nepotvrdí editaci, má jistotu, že neprovedl žádné změny v databázi.

Ukázkový formulář nalezneme na obrázku 21 .

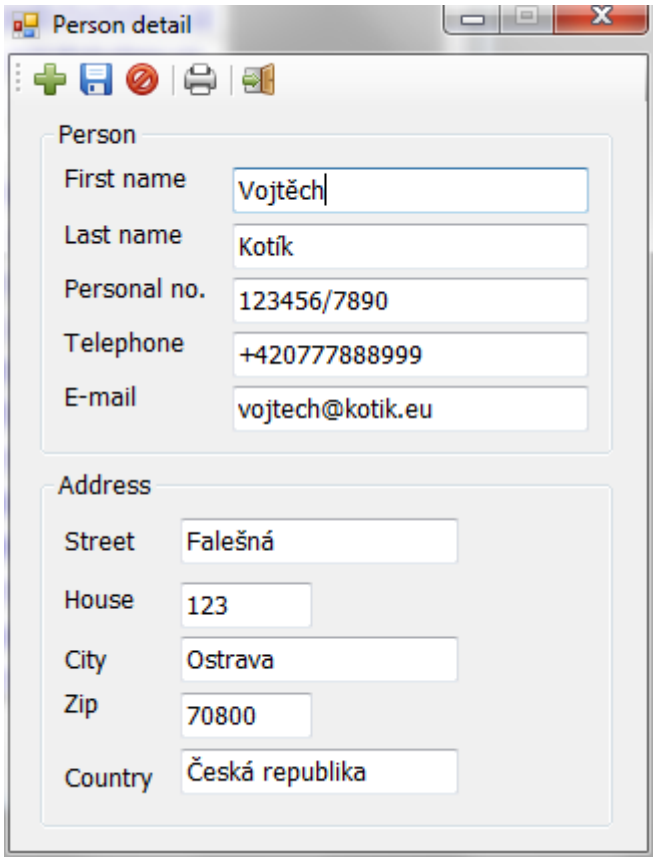
5.3 FormList

Pro editace číselníků by bylo vzhledem k malému počtu záznamů zbytečně komplikované používat kombinace přehledu a detailu. Z tohoto důvodu je použita ještě speciální šablona pro tyto případy. Formuláře z ní dědící obsahují komponentu DataGridView, která umožňuje jednoduše editovat záznamy po dvojkliku myší. Po ukončení editace klepnutím na tlačítko OK dochází k okamžité aktualizaci dat v databázi.

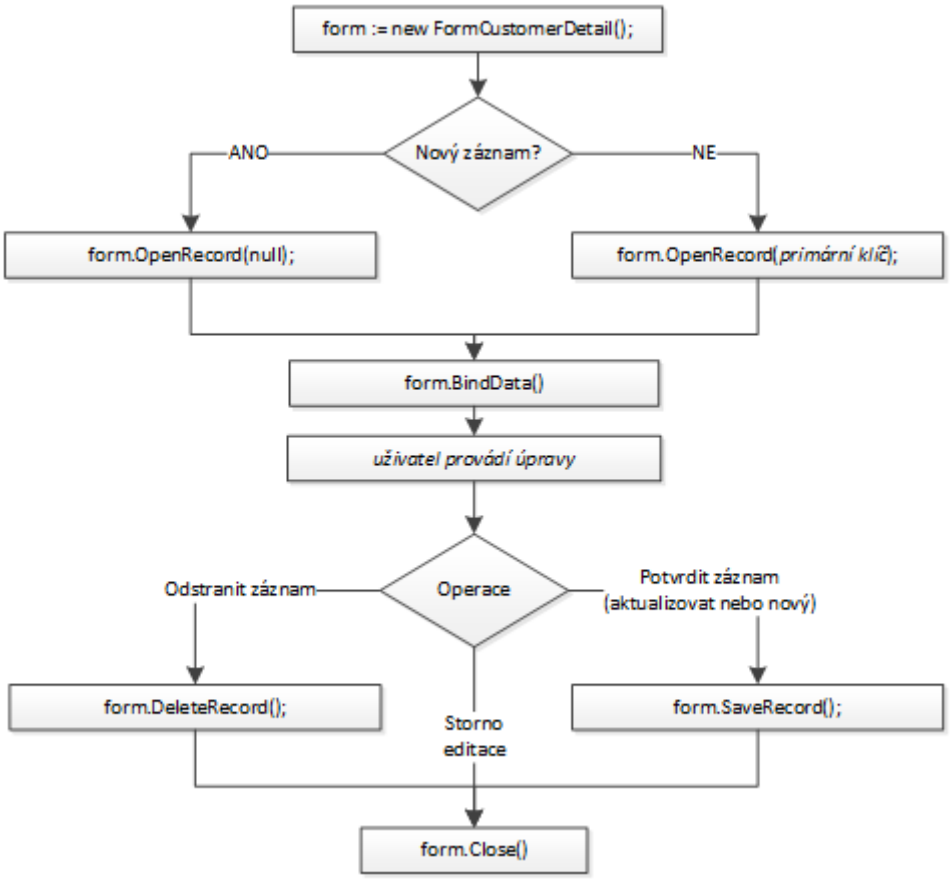
Ukázkový formulář nalezneme na obrázku 23 .

5.4 FormDialog

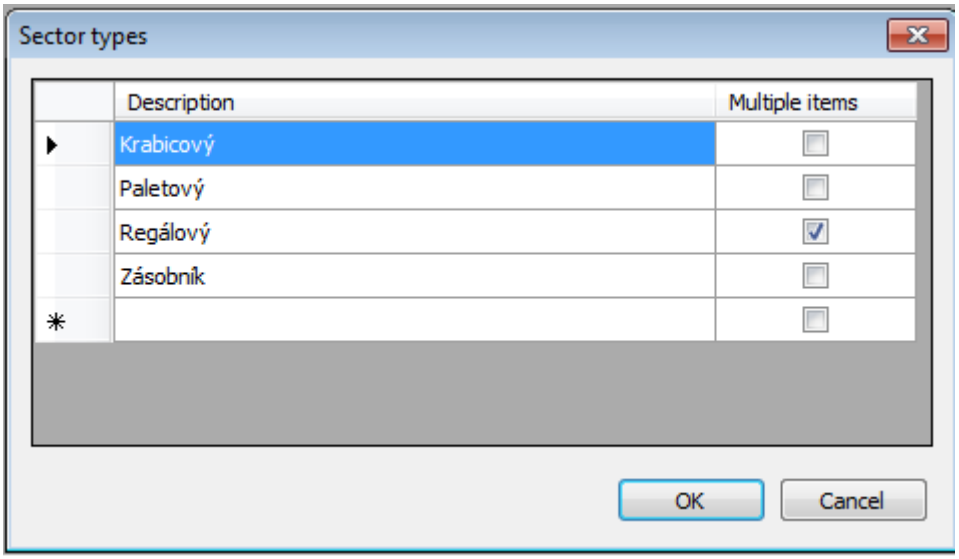
Obecná šablona pro dialogová okna, která vyžadují zadání či výběr dat uživatelem. V systému jsou v hojné míře použity pro výběry konkrétní položky ze seznamu, jako je přiřazení zákazníka objednávce, nebo nastavení sektoru a výběr produktu při naskladnění zboží.



Obrázek 21: Ukázkový FormDetail

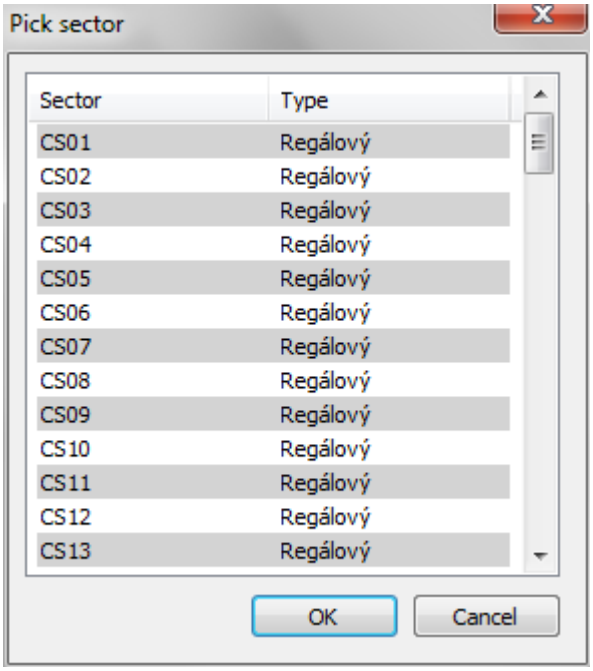


Obrázek 22: Použití FormDetail



Obrázek 23: Ukázkový FormList

Ukázkový formulář je na obrázku 24 .

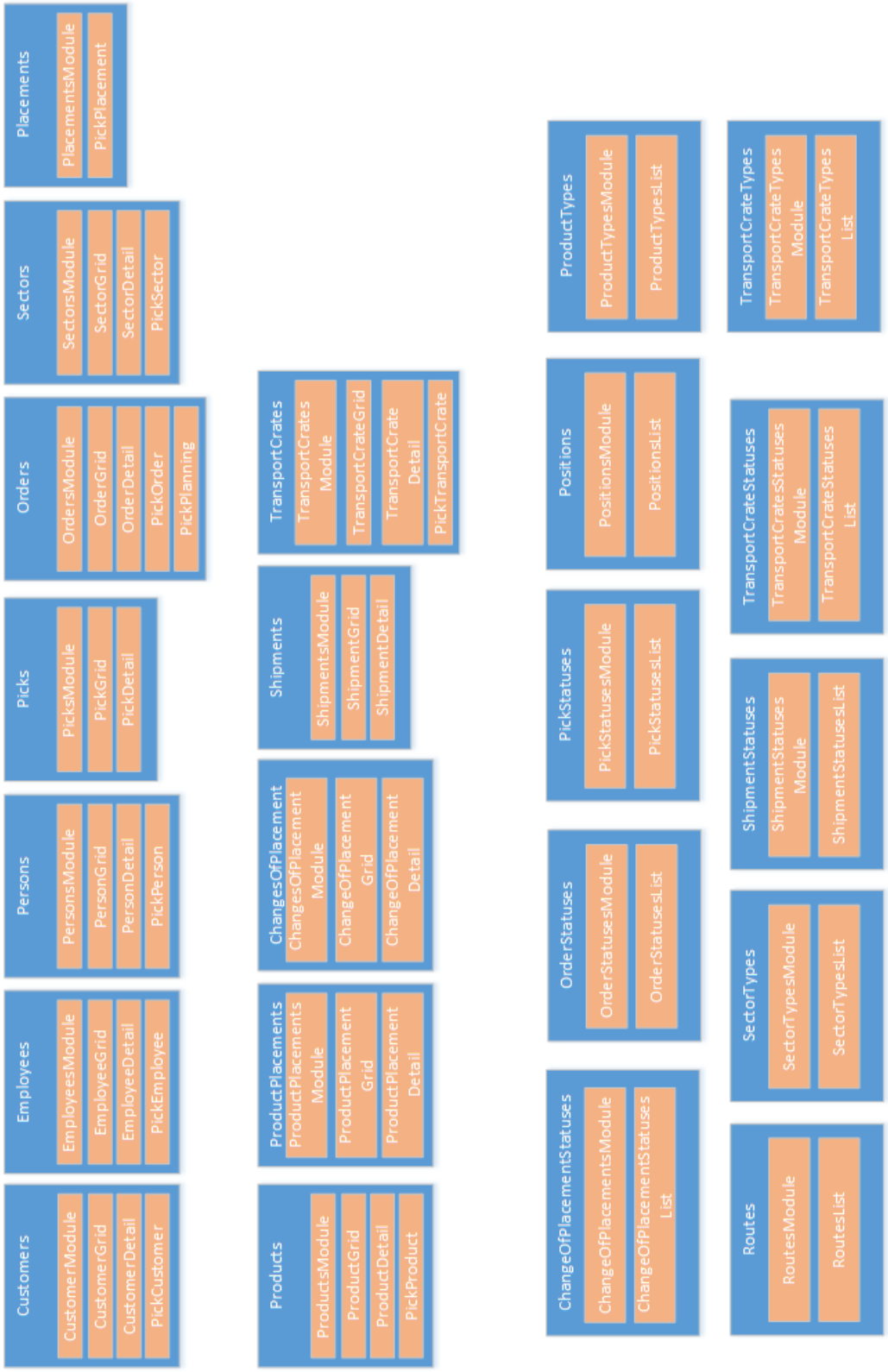


Obrázek 24: Ukázkový FormDialog

5.5 Module

Třída tvořící přístupový bod k danému modulu. Pomocí metody *Run* dochází ke spuštění modulu, vytvoření a zobrazení příslušného formuláře s přehledem.

Seznam všech implementovaných modulů a jejich součástí je k nalezení na obrázku 25 .



Obrázek 25: Moduly a jejich součásti

6 Návrh komponenty pro editaci rozložení skladu

Tato kapitola je věnována návrhu programové komponenty, která umožní uživatelsky přívětivou cestou definovat rozložení skladu. Jako matematický model pro tuto situaci se přímo nabízí neorientovaný ohodnocený graf, jehož vrcholy představují sektory ve skladu a hrany délky cest mezi těmito sektory. Kromě použití vrcholů pro reprezentaci sektorů se v grafu budou nacházet také pomocné vrcholy pro pohodlnější editaci (chodby) a několik málo vrcholů reprezentujících startovací pozice skladníků (pick stations). Představa způsobu editace vychází z obrázku 26 . Komponenta by měla umět zobrazit půdorys skladu. Půdorys může být načten ze standardního rastrového obrazového souboru jako např. JPG nebo PNG. Uživatel poté myší umístí na vhodné pozice vrcholy grafu a u každého vrcholu vybere, zda se bude jednat o sektor, kde skladník zahajuje svou cestu (pick station), pomocný vrchol nebo standardní sektor, ve kterém se nachází sortiment. Poté uživatel vybere vždy dvojici vrcholů a propojí je hranou, přičemž u hrany uvede její hodnotu (fyzickou vzdálenost vrcholů) např. v metrech.

Reprezentace grafu

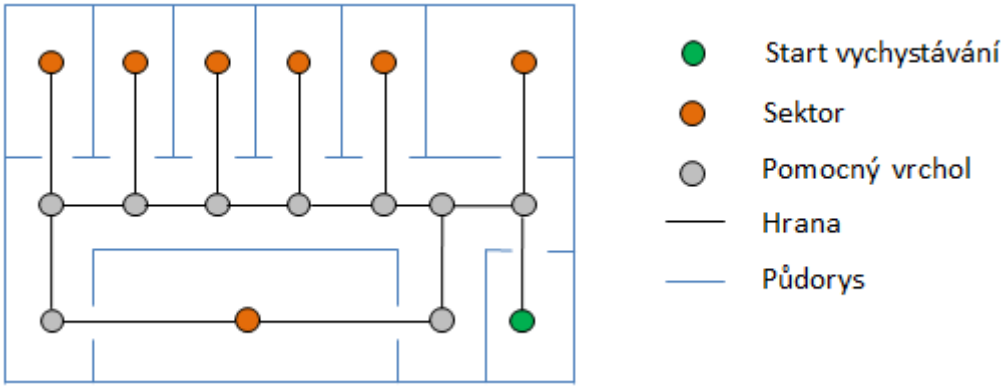
Pro reprezentaci grafu skladu byly vytvořeny 3 třídy: *StoreGraph*, *StoreVertex* a *StoreEdge* pro reprezentaci grafu jako celku, vrcholů a hran. UML diagram této reprezentace můžeme nalézt na obrázku 27 . Vidíme, že samotná třída *StoreGraph* obsahuje kromě seznamu vrcholů a hran také metodu pro vytvoření vrcholu *CreateVertex* a metodu *CreateEdge* pro vytvoření hrany mezi dvěma danými vrcholy. Tyto dvě metody kromě instanciování tříd *StoreVertex* a *StoreEdge* současně zařadí vzniklé objekty do instance grafu. Metoda *CreateVertex* navíc přiděluje vrcholům unikátní čísla (vertexId), která umožňují serializaci a deserializaci grafu. Serializace a deserializace grafu je zajištěna metodami *SaveToXml* a *LoadFromXml* s parametrem typu *XmlNode*. Celý graf je tedy možné uložit do XML dokumentu, který lze následně uložit do databáze nebo zpětně načíst.
U vrcholu (třída *StoreVertex*) je navíc uveden typ vrcholu (*sektor*, *chodba* nebo *pick station*) a 2-D souřadnice X, Y. U hrany je navíc uvedena její délka.

Komponenta

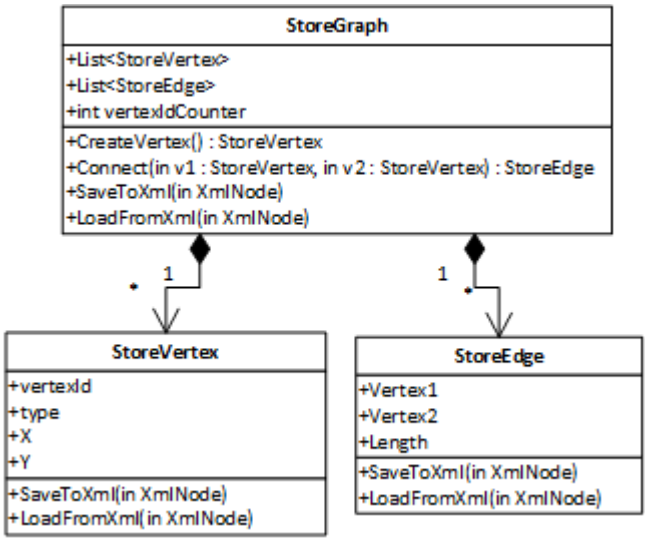
Programová komponenta byla vytvořena děděním ze standardní třídy .NET Frameworku *UserControl*. Účelem komponenty je editovat instanci třídy *StoreGraph* popsané v předchozí kapitole. Stěžejní je metoda *Render*, která na předaný argument typu *Graphics* vykreslí celý graf – vrcholy jako kružnice o průměru 6px (pixelů) a hrany jako úsečky. Před vykreslením grafu jako takového je na pozadí komponenty navíc vykreslen načtený obrazový soubor, který reprezentuje podklad (půdorys) skladu, pro snadnější zadávání nových vrcholů a hran. Uživatel provádí tři operace:

1. Výběr vrcholu nebo hrany
2. Vytvoření vrcholu
3. Vytvoření hrany

Grafické rozhraní pro editaci grafu skladu nalezneme na obrázku 28 .



Obrázek 26: Ukázka grafu



Obrázek 27: Třídy pro reprezentaci grafu

6.1 Výběr vrcholu nebo hrany

Součástí třídy komponenty jsou instanční proměnné reprezentující aktuálně vybraný vrchol nebo hranu. Při vykreslování jsou vybraný vrchol nebo hrana znázorněny odlišnou barvou. Pro samotný výběr vrcholu nebo hrany je potřeba základních znalostí analytické geometrie.

Vrchol

Při stisknutí tlačítka myši zjistíme aktuální souřadnice x_m a y_m. Poté spočítáme vzdálenost d do každého vrcholu v jednoduše pomocí Pythagorovy věty $d = \sqrt{(x_v - x_m)^2 + (y_v - y_m)^2}$. Je-li tato vzdálenost pro určitý vrchol menší než stanovený limit (5 px), je vrchol v označen jako vybraný.

Hrana

Uživatel opět stiskne tlačítko myši na souřadnicích x_m a y_m. Každá hrana definuje obdélník (tzv. *bounding rect*) jehož protější vrcholy jsou tvořeny vrcholy grafu. Nejprve vyfiltrujeme pouze ty hrany, jejichž *bounding rect* zahrnuje souřadnice x_m , y_m. Pro každou hranu, která vyhoví této podmínce, následně spočítáme vzdálenost d bodu x_m a y_m od přímky definované dvěma body, které odpovídají souřadnicím vrcholů hrany. Vzdálenost bodu od parametricky dané přímky ve tvaru $ax + by + c = 0$ spočítáme podle známého vztahu [2] $d = \frac{\|ax_m + by_m + c\|}{\sqrt{a^2+b^2}}$. Převodem přímky dané dvěma body na parametrické vyjádření dojdeme ke vztahům, které umožní určit konkrétní hodnoty a, b a c.

$$
\begin{aligned}
a &= y_1 - y_2 \\
b &= x_2 - x_1 \\
c &= (y_2 - y_1)x_1 + (x_1 - x_2)y_1
\end{aligned}
$$

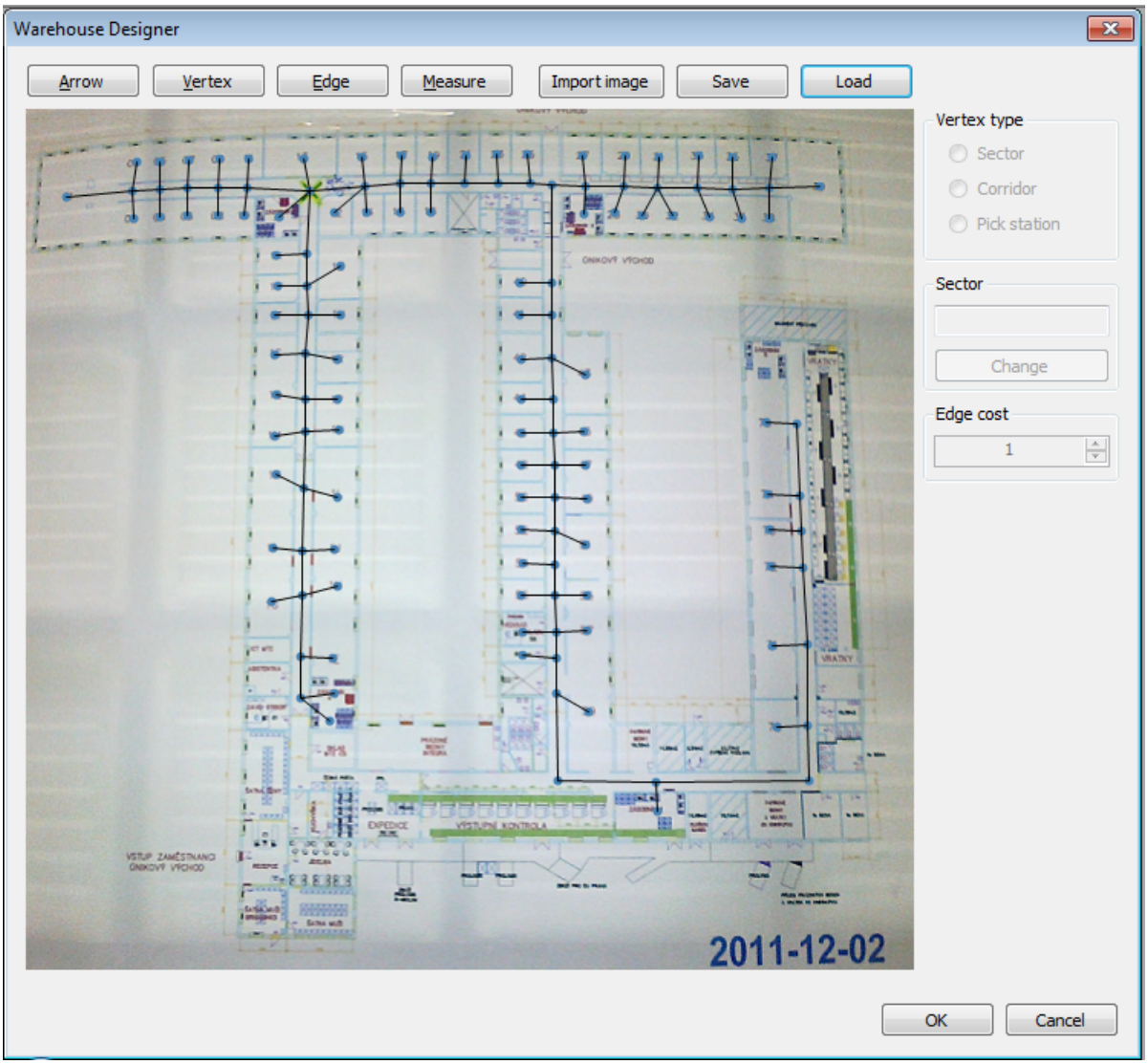
Nakonec opět zvolíme vhodný limit vzdálenosti, od kterého považujeme hranu za vybranou.

6.2 Vytvoření vrcholu

Vytvoření vrcholu je velmi snadná záležitost. Na instanci třídy *StoreGraph* pouze zavoláme metodu *CreateVertex* a vytvořenému vrcholu nastavíme aktuální souřadnice kurzoru myši.

6.3 Vytvoření hrany

Při vytváření hrany předpokládáme, že je aktuálně jeden vrchol v_1 označený jako vybraný. Při stisku tlačítka myši zjistíme, zda se v blízkosti aktuální pozice kurzoru nachází jiný vrchol v_2. Pokud ano, vytvoříme hranu mezi v_1 a v_2.



Obrázek 28: Editor grafu skladu

7 Návrh algoritmu pro optimální cestu skladníka

Tato část práce je věnována návrhu algoritmu, který by měl optimalizovat cestu skladníka při vychystávání objednávky. Detailní popis, co představuje objednávka a vychystávání, byl uveden dříve v kapitole 2.1, nicméně pro úplný popis problému si zde shrneme nejdůležitější fakta.
Objednávka se skládá z položek, které jsou vždy definovány typem sortimentu a počtem kusů. Skladník vychystává najednou vždy jen jedinou objednávku, jejíž obsah ale bývá rozsáhlý. Proto je potřeba rozdělit položky objednávky do tzv. picků a každý pick se poté vychystává. Vychystáváním se rozumí to, že skladník s vozíkem projde skladem (velká hala), navštěvuje jednotlivé sektory a odebírá z nich knižní tituly v požadovaném množství. Abychom mohli minimalizovat délku trasy skladníka, musí v našem systému existovat definice pozic sektorů na skladu a tras mezi nimi. Tato definice by měla mít ideálně podobu neorientovaného ohodnoceného grafu. Z toho důvodu bylo nutné navrhnout speciální modul, který by tento graf umožňoval nadefinovat pro libovolný sklad (viz kapitola 6).

7.1 Vstup

Vstupy do navrhovaného algoritmu jsou následující:

1. Ohodnocený neorientovaný graf, kde vrcholy odpovídající sektorům lze rozdělit do tří disjunktních množin:
   (a) Sektory
   (b) Chodba
   (c) Startovací místo vychystávání (tzv. *Pick station*)
2. Objednávka skládající se z položek, kde u každé položky je uvedeno:
   (a) Sortiment (cizí klíč do tabulky *Product*)
   (b) Množství
3. Kapacita přepravní krabice
4. Informace o dostupném množství sortimentu na jednotlivých sektorech
5. Sektor typu *Pick station*

7.2 Výstup

1. Rozdělení položek objednávky na jednotlivé picky
2. U každého picku:
   (a) Startovací místo
   (b) Cesta procházející určitými sektory, ze kterých je potřeba vychystat sortiment v určitém množství.

7.3 Algoritmus

Na první pohled souvisí tento problém se známým problémem obchodního cestujícího. Tj. zjednodušeně potřebujeme navštívit všechna města (v našem případě sektory) za nejkratší trasu. Jak je známo, tento problém patří mezi tzv. NP-těžké problémy, pro které neexistuje algoritmus, který by je řešil v polynomiálním čase. Náš algoritmus tedy nebude nacházet nejoptimálnější výsledek (tzn. takový, aby se skladník nejméně nachodil), ale bude se snažit nalézt v rámci možnosti vhodnou cestu. Dále zde provedeme malé zjednodušení – skladník se vždy bude vracet na místo odkud vycházet. Nemůže tedy dojít na jiný sektor typu *Pick station*.

7.4 Rozdělení objednávky na picky

Prvním krokem algoritmu je rozdělení objednávky na jednotlivé picky. Máme vybránu přepravní krabici o určité kapacitě a je potřeba rozdělit položky objednávky tak, aby se nám vždy vešly do této krabice. V tuto chvíli zanedbáme, že objem krabice nikdy nemůže být 100% využitý a zkrátka budeme předpokládat, že např. do krabice o objemu 20000 cm^3 se nám vejde 50 knih o objemu 400 cm^3. Rozdělení objednávky na picky bude probíhat podle následujícího postupu:

1. Inicializujeme pomyslnou krabici o určitém objemu.
2. Setřídíme položky objednávky podle jejich celkového objemu sestupně.
3. Odebereme první položku z objednávky a umístíme ji do krabice. Současně kontrolujeme, zda jsme nepřekročili stanovený objem.
   (a) Pokud je objem překročen, umístíme do krabice pouze takové množství položky, jaké se tam vejde, zbytek vrátíme do objednávky. Přitom uzavíráme pick a vracíme se k bodu 1.
   (b) Pokud jsme objem nepřekročili, pokračujeme krokem 3.
   (c) Je-li objednávka prázdná, uzavřeme poslední pick a ukončíme algoritmus.

Uvedený algoritmus se snaží minimalizovat počet titulů, které je potřeba vychystat během jednoho picku. Čím méně různých titulů, tím méně sektorů bude pravděpodobně potřeba projít. Na druhou stranu v určitých případech se algoritmus nebude chovat nejlépe – není zohledněno, že např. 3 různé tituly se klidně mohou nacházet na jednom sektoru. Algoritmus samozřejmě počítá s tím, že neexistuje kniha, která by se samotná nevešla do krabice, jinak by mohlo dojít k zacyklení, protože objednávka by se nikdy nevyprázdnila.

7.5 Varianty navštívených sektorů

Nyní se problém přesouvá do plánování cesty pro každý pick zvlášť. V tuto chvíli máme dány položky, jejich množství a startovací místo (pick station). Musíme projít sklad tak, abychom vychystali všechny potřebné položky a přitom abychom urazili co nejkratší cestu.

Jak již bylo uvedeno dříve, vždy se vracíme zpět do startovacího místa (pick station). Situace jednoho picku je ilustrována tabulkou 1. Nyní je potřeba nahlédnout do databáze a zjistit, jaké množství požadovaného sortimentu se nachází na jakých sektorech. Je zde nutno vyřešit malý implementační problém – množství sortimentu se ve skladu uvádí ke skladovacím pozicím, které se nacházejí v sektorech – zde si však pomůžeme vhodným SQL dotazem s použitím agregační funkce SUM.

Sortiment	**Množství**
Skripta LAIT	20
Skripta MAIT	10
Skripta DIM	10

Tabulka 1: ukázkový Pick

V tabulce 2 můžeme vidět, jak může vypadat situace ve skladu pro požadované tituly.

Sortiment	**Sektor**	**Množství**
Skripta LAIT	A	100
Skripta LAIT	B	50
Skripta LAIT	C	15
Skripta LAIT	D	15
Skripta LAIT	E	5
Skripta LAIT	F	5
Skripta LAIT	G	10
Skripta MAIT	A	15
Skripta MAIT	D	5
Skripta MAIT	H	5
Skripta DIM	I	15
Skripta DIM	J	20

Tabulka 2: stav skladu

Nyní bychom měli ke každé položce picku (tzn. ke každému řádku v tabulce 1) najít kombinace sektorů, které musíme navštívit a kolik z těchto sektorů máme odebrat knih.

Sortiment	**Množství**	**Sektory**
Skripta LAIT	20	(A:20), (B:20), (C:15 D:5), (C:15 E:5), (C:15 F:5), (C:15 G:5), (D:15 E:5), (D:15 F:5), (D:15 G:5), (E:5, F:5, G:10)
Skripta MAIT	10	(A: 10), (D:5 H:5)
Skripta DIM	10	(I:10), (J:10)

Tabulka 3: varianty odběru ze skladu

V tabulce 3 můžeme pro každou položku v picku vidět, jakou kombinaci sektorů musíme navštívit, abychom vychystali potřebné množství. Např. u Skritpt LAIT můžeme zajít jen do sektorů A nebo B, které nám samy o sobě pokryjí celé požadované množsvtí. Jinou možností je navštívit zároveň sektor C i E, protože samostatně tyto sektory sice požadované množství neposkytnou, ale dohromady ano.
Je potřeba si uvědomit, že ne vždy je vhodné řídit se pravidlem – čím méně sektorů u jedné položky, tím lépe. U kombinací více sektorů se nám může stát, že nám tyto sektory pokryjí jinou položku v picku.

7.6 Optimální výběr navštívených sektorů

Nyní je nutné projít výsledky předchozího kroku (tzn. poslední sloupec v tabulce 3) a provést kartézský součin jednotlivých variant. Takto obdržíme všechny možné varianty, jaká skladová umístění musíme během cesty navštívit. Tzn. výsledek této operace na ukázkovém příkladu z tabulky 3 vypadá následovně:

1. Sektor A – 20 x Skripta LAIT; Sektor A – 10 x Skripta MAIT; Sektor I – 10 x Skripta DIM
2. Sektor B – 20 x Skripta LAIT; Sektor A – 10 x Skripta MAIT; Sektor I – 10 x Skripta DIM
3. Sektor C – 15 x Skripta LAIT; Sektor E – 5 x Skripta LAIT; Sektor A – 10 x Skripta MAIT; Sektor I – 10 x Skripta DIM
4. ...

Nyní nás zajímají už pouze množiny sektorů, které se vyskytují v jednotlivých vyjmenovaných kombinacích. Tzn. např. u první vyjmenované kombinace, uvažujeme sektor A pouze jednou (tím lépe, že z tohoto jednoho sektoru odebíráme dva tituly).
V tuto chvíli již skutečně řešíme standardní problém obchodního cestujícího, kdy máme dánu množinu vrcholů v grafu, které musíme postupně navštívit nejkratší cestou tak, abychom se nakonec vrátili do vrcholu, ze kterého jsme vycházeli. Jak již bylo uvedeno, jedná se o problém, pro který neexistuje algoritmus pracující s polynomiální časovou složitostí (v závislosti na počtu vrcholů a hran). Existují však algoritmy, jejichž výsledek se blíží optimální cestě.
V našem postupu jsme použili jednoduchou metriku, která alespoň odhadne, jaká kombinace navštívených sektorů je vhodná. Tato metrika uvažuje součet vzdáleností mezi každými dvěma sektory (včetně startovního sektoru). Množina sektorů, která má tento součet nejmenší představuje celý výsledek našeho algoritmu, přičemž při výpisu uspořádáme sektory podle vzdálenosti od počátečního sektoru (pick station). Pro výpočet vzdáleností mezi dvěma vrcholy byl naimplementován standardní Dijkstrův algoritmus.

Dijkstrův algoritmus

Na webu [1] je popsán takto:

Na Dijkstrův algoritmus lze pohlížet jako na zobecněné prohledávání grafu do šířky, při kterém se vlna nešíří na základě počtu hran od zdroje, ale vzdálenosti od zdroje (ve smyslu váhy hran). Tato vlna proto zpracovává jen ty uzly, k nimž již byla nalezena nejkratší cesta.

Dijkstrův algoritmus si uchovává všechny uzly v prioritní frontě řazené dle vzdálenosti od zdroje - v první iteraci má pouze zdroj vzdálenost 0, všechny ostatní uzly nekonečno. Algoritmus v každém svém kroku vybere z fronty uzel s nejvyšší prioritou (nejnižší vzdáleností od již zpracované části) a zařadí jej mezi zpracované uzly. Poté projde všechny jeho dosud nezpracované potomky, přidá je do fronty, nejsou-li tam již obsaženi, a ověří, zda-li nejsou blíže zdroji, než byli před zařazením právě vybraného uzlu mezi zpracované. To znamená, že pro všechny potomky ověřuje: $vzdalenost_{zpracovavany} + delkaHrany_{zpracovavany,\, potomek} < vzdalenost_{potomek}$

Pokud nerovnost platí, tak danému potomkovi nastaví novou vzdálenost a označí za jeho předka zpracovávaný uzel. Po průchodu přes všechny potomky algoritmus vybere z fronty uzel s nejvyšší prioritou a celý krok opakuje.

Algorimus terminuje v okamžiku, kdy jsou zpracovány všechny uzly (prioritní fronta je prázdná).

Dijkstrův algoritmus je použitelný jen tehdy, obsahuje-li graf pouze nezáporně ohodnocené hrany - v opačném případě není schopen garantovat, že při zpracování uzlu byla již nalezena nejkratší možná cesta.

8 Srovnání s existujícím systémem

Hlavní výhodou oproti stávajícímu informačnímu systému je vytvoření algoritmu zpracovávajícímu přijaté objednávky. V současném systému implementován není a tak dochází k situacím, kdy je např. vygenerován pick pro až několika set násobky maximální kapacity bedny, do které má být vychystáváno. To způsobuje průtahy při průchodu objednávky systémem a někdy dochází i k poškození vychystávaného sortimentu, když se méně zkušený chystač snaží do bedny umístit vše, co by tam správně mělo být.
Tento problém jsme úspěšně vyřešili a v nově vytvořeném informačním systému již k němu nedochází.

9 Závěr

Cíle této bakalářské práce bylo vytvoření informačního systém mapující procesy knižního skladu včetně efektivního plánování cesty skladníka při vychystávání zboží.
Tento nápad a znalost popisovaných procesů jsem získal na základě osobní pracovní zkušenosti ve skladu firmy Booklogistics, kde jsem prošel všechny pozice od řadového chystače až po vedoucího skladu a měl jsem jedinečnou možnost nahlédnout do skladových procesů a informačního systému z mnoha rovin.
Problematika provozu knižního skladu je velmi rozsáhlá. Vytvořený informační systém není schopen pokrýt veškerou problematiku, nicméně může sloužit jako základ pro vývoj rozsáhlého a kompletního informačního systému pro správu knižního skladu. Rovněž algoritmus plánování cesty skladníka je možno do budoucna rozšířit a vylepšit jeho výkonnost.

10 Reference

[1] *Dijkstrův algoritmus. Algoritmy.net [online]. [cit. 2014-05-06]. Dostupné z: http://www.algoritmy.net/article/5108/Dijkstruv-algoritmus*

[2] *Vzdálenost bodu od přímky. Matematika.cz [online]. [cit. 2014-05-06]. Dostupné z: http://www.matematika.cz/vzdalenost-bod-primka*

11 Přílohy

Příloha 1

Kompletní schéma databáze (DVD)

Příloha 2

Zdrojové kódy prototypu databáze (DVD)